Introduzione al C++

Connessione ad internet

Istruzioni per la connessione internet:

- una volta connessi, aprire un browser (firefox)

- in **Modifica/preferenze/avanzate/rete/impostazioni**

- attivare la modalità "**configurazione manuale del proxy**"

ed inserire "**proxyu9.lib.unimib.it**" - porta **8080**

- attivare "**utilizza questo proxy per ogni protocollo**"

- confermare tutto e tornare alla pagina del browser

- se viene richiesto di inserire utente e password, mettete le vostre

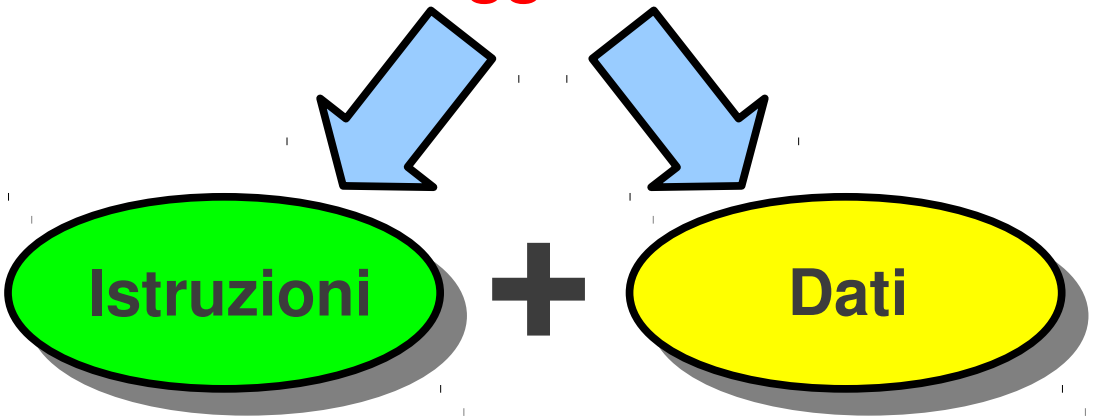
(utente è : lib\bi01matricola)

Documentazione

- http://lab-info.blogspot.com/
  - per scambiare informazioni, suggerimenti, soluzioni
- http://lab-info-q.blogspot.com/
  - per dettagli ed approfondimenti
- Per contattarci :
  - luigi.zanotti@mib.infn.it
  - davide.chiesa@mib.infn.it
  - marco.pizzichemi@gmail.com
  - n.divara@campus.unimib.it

Programmazione a oggetti

- Si **scompone** un problema nelle parti che lo costituiscono
- Ciascuna parte diventa un **oggetto**



- **Complessità** di programmazione notevolmente ridotta
- Ulteriori **vantaggi** dall'implementazione di:
  - Incapsulamento
  - Polimorfismo
  - Ereditarietà

Sorgente ed eseguibile

E' necessario avere ben presente la distinzione tra:

- **Codice Sorgente**: versione “leggibile dagli esseri umani”
- **Codice Eseguibile**: versione che il pc può leggere ed eseguire

Il compito di trasformare la prima nella seconda è assolto dal **compilatore**, nel nostro caso c++ (o g++)

- Come scriviamo il Codice Sorgente?

  Editor di testo → (Gedit, Emacs, Kate, Vim...)

- Come lo compiliamo?

  c++ -o <nome eseguibile> <nome_sorgente>

Primo programma in C++

- Scriviamo il seguente programma:

```
/*  Programma #1 - Un primo progromma in C++.
    Scrivete questo programma in un file,
    compilatelo e avviatelo
*/
#include <iostream>

int main()
{
  std::cout << "This is my first C++ program.";
  return 0;
}
```

- Compiliamo ed avviamo da terminale, così:

```
marco@barattolo:2011$ c++ -o esempio_1_00 esempio_1_00.cpp
marco@barattolo:2011$ ./esempio_1_00
This is my first C++ program.marco@barattolo:2011$
```

Analizziamo il codice sorgente

Commenti ignorati dal compilatore, molto utili per il programmatore. Altro metodo, usare //, ma vale solo per una riga!

Inclusione di header (libreria): molte funzioni di cui avremo bisogno sono contenute in librerie messe a disposizione dal compilatore

```
/*  Programma #1 - Un primo progromma in C++.
    Scrivete questo programma in un file,
    compilatelo e avviatelo
*/
#include <iostream>

int main()
{
  std::cout << "This is my first C++ program.";
  return 0;
}
```

Dichiarazione della funzione principale, obbligatoria. Ogni programma inizia chiamando *main*. Preceduto da dichiarazione del tipo di valore ritornato (*int* in questo caso)

- std::cout → chiamata a funzione *cout* (console output) contenuta nelle librerie standard *std*
- << → operatore di output
- "This... " → stringa di testo
- ; → tutte le istruzioni terminate dal punto e virgola

Termina il programma e ritorna il valore 0 (intero) al processo che ha invocato il programma stesso (tipicamente il sistema operativo)

Un programma più completo...

- Provate il seguente programma:

```
#include<iostream>

int main (int numArg, char *listArg[])
{
  std::cout << "nome del programma: "
            << listArg[0] << std::endl ;

  int num;
  std::cout << "inserisci un numero "
            << std::endl ;
  std::cin  >> num ;
  std::cout << "il numero inserito è: "
            << num << std::endl ;

  return 0 ;
}
```

- Unici argomenti possibili per la funzione *main* sono:
  - numero di argomenti passati da riga di comando (int)
  - lista degli argomenti (char)
- Operatore di input >>
- Funzione *endl*

- Qual è il primo argomento passato al *main* dal SO?
- Provate a passare altri argomenti al programma e farvi restituire il numero totale di argomenti

Le variabili

- Quantità di interesse si gestiscono come variabili

```
int num1 = 0 ;
int num2 = 3 ;
int somma = num1 + num2 ;
std::cout << "somma: "
          << somma << "\n" ;
float razionale = 3.1416 ;
double razionale2 = 1.4142 ;
char lettera = 'a' ;
bool condizione = true ;
```

```
int vettore[10] ;
for (int i = 0 ; i < 10 ; ++i)
  {
    vettore[i] = i * 2 ;
  }
```

- Diversi oggetti sono gestiti da diversi **tipi** (int, float, char...)
- Una variabile e' **l'istanza di un tipo** (num1 e' una istanza di int)
- Le normali operazioni che ci si aspetta sono definite sui tipi
- Si possono creare vettori di un tipo di variabili, chiamati **array**
- La lista degli oggetti di un array occupa una zona contigua della memoria
- La dimensione di un array e' una costante

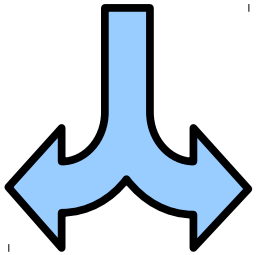
Le variabili

Type	Bit Width	Common Range
char	8	–128 to 127
unsigned char	8	0 to 255
signed char	8	–128 to 127
int	16	–32,768 to 32,767
unsigned int	16	0 to 65,535
signed int	16	–32,768 to 32,767
short int	16	same as int
unsigned short int	16	same as unsigned int
signed short int	16	same as short int
long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
signed long int	32	–2,147,483,648 to 2,147,483,647
float	32	3.4E–38 to 3.4E+38
double	64	1.7E–308 to 1.7E+308
long double	80	3.4E–4932 to 1.1E+4932
bool	N/A	true or false
wchar_t	16	0 to 65,535

Gli operatori

- Avete a disposizione diversi operatori definiti in c++:
  - operatori **aritmetici**: +, -, *, /, %
  - operatori di **incremento e decremento**: ++, --
  - operatori **logici e relazionali**: >, >=, <, <=, ==, !=, &&, ||, !
- Fate sempre attenzione alla **precedenza** tra operatori...

Più alta	**++ --**
	/ * %
Più bassa	**+ -**



Più alta	**!**
	> >= < <=
	== !=
	&&
Più bassa	**||**

- ...e alla **conversione** tra tipi delle espressioni aritmetiche → in una operazione tra tipi diversi, tutti gli operandi vengono convertiti al tipo "più grande" **prima** di compiere l'operazione

Suggerimenti e esercizi

- Fate uso delle parentesi nelle espressioni matematiche!
- Potete utilizzare, se necessario, un ***cast*** tra tipi. La sintassi generica è *(tipo) espressione*
  - Ad esempio *(float) x / 2* vi restituirà un tipo float come risultato

- **Esercizio**: scrivete un programma che, letti due numeri interi da terminale, restituisca il loro rapporto
- **Esercizio**: scrivete un programma che, lette due variabili booleane da terminale, restituisca il risultato delle operazioni AND, OR e XOR
  (suggerimento: il tipo *bool* in c++ accetta solo due valori, *true* e *false*, ma questi sono del tutto equivalenti a 1 ed 0)

Strutture di controllo: for

- Un'operazione da ripetere molte volte puo' essere automatizzata:

```
for (int index = 0 ; index < numArg ; ++index)
  {
    std::cout << index << "-esimo argomento: "
              << listArg[index]
              << std::endl ;
  }
```

- Le istruzioni nello scope vengono eseguite finche' **index < numArg**
- La variabile **index** esiste soltanto nello scope del ciclo
- Il ciclo viene impostato **definendo** index, stabilendo la **condizione di uscita** e dettando l'istruzione di **incremento** della variabile di controllo

Strutture di controllo: while

- Il **for** non e' l'unico modo per realizzare un ciclo:

```
int index = 0 ;
while (index < numArg)
  {
    std::cout << index << "-esimo argomento: "
              << listArg[index]
              << std::endl ;
    ++index ;
  }
```

- Le istruzioni nello scope vengono eseguite finche' **index < numArg**
- La variabile **index** esiste anche al di fuori del ciclo
- E' necessario definire correttamente la variabile di controllo prima dell'inizio del ciclo

Strutture di controllo: if

- Eseguire istruzioni solo se verificata una condizione

```
if (numArg > 1)
  {
    std::cout << "primo argomento opzionale: "
              << listArg[1] << std::endl ;
  } else {
    std::cout << "non ci sono argomenti opzionali"
              << std::endl ;
  }
```

- Se l'espressione di controllo è vera, viene eseguita la sequenza di istruzioni nel primo *scope* (le parentesi {})
- Altrimenti viene eseguita la sequenza dopo ***else***
- La presenza di *else* non è obbligatoria!

Strutture di controllo: switch

- Se le possibilita' non sono due (vero/falso), ma **molte**

```
switch (x)
{
  case 1:
    std::cout << "x vale 1";
    break;

  case 2:
    std::cout << "x vale 2";
    break;

  default:
    std::cout << "x non vale ne 1 ne 2";
}
```

- La variabile **x** e' gia' definita
- Per ogni possibile valore c'e' un'istruzione da eseguire
- L'istruzione **break** e' necessaria per terminare subito lo **switch**
- Il **default** e' l'istruzione che viene eseguita se nessuno dei casi precedenti e' vero

Operazioni matematiche

- Operatori matematici non presenti nelle librerie standard sono forniti dalle librerie matematiche

```
#include<iostream>
#include<cmath>

int main (int numArg, char *listArg[])
{
  double due = 2 ;
  double radice_due = sqrt (due) ;
  std::cout << radice_due << std::endl ;
}
```

- E' necessario includere la libreria **cmath** per avere a disposizione le funzioni necessarie
- Le funzioni matematiche sono chiamate nel programma

Esercizi

- Scrivere un programma che scrive a terminale
  - la radice quadrata di 2, il cubo di 2 il seno di pigreco/4

Esercizi

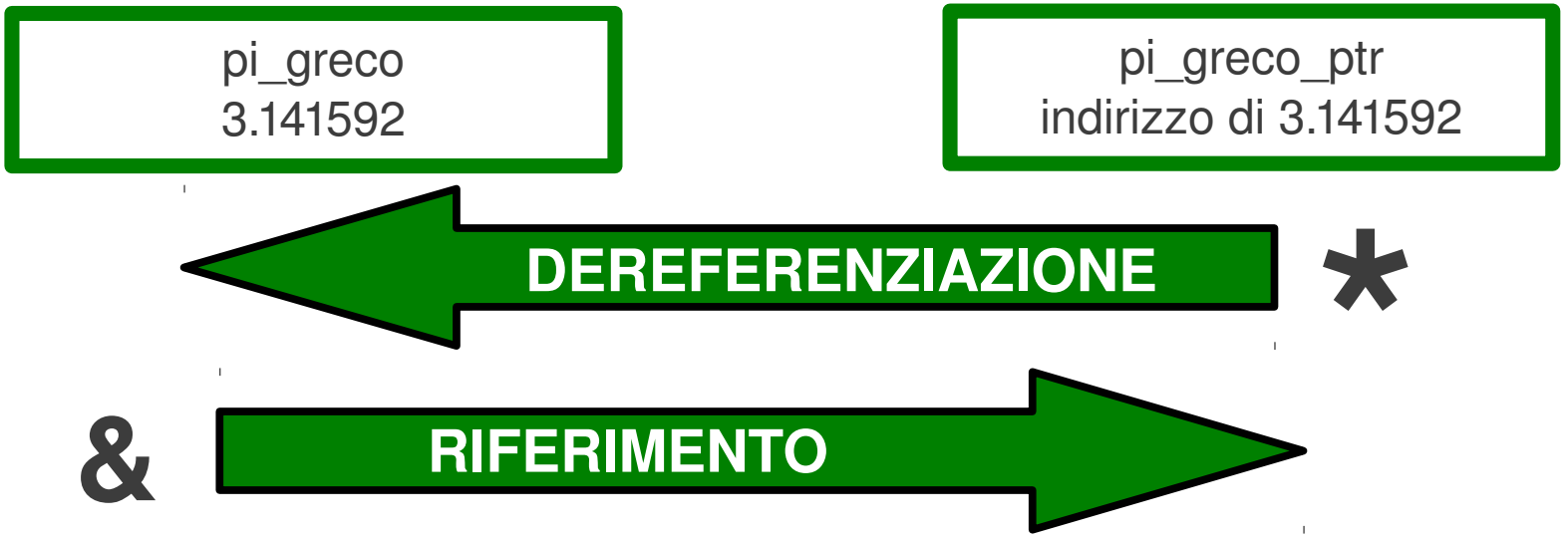
- Scrivere un programma che scrive a terminale
  - la radice quadrata di 2, il cubo di 2 il seno di pigreco/4
- Scrivere un programma che:
  - controlli se uno o più argomenti opzionali sono passati al main al momento dell'avvio (e in caso contrario, avvisi l'utente)
  - scriva l'intera lista di argomenti opzionali passati al main
  - chieda all'utente di inserire un intero a scelta tra 1 e 2, e restituisca a terminale il valore inserito, o un messaggio di errore in caso di inserimento di altri interi

Esercizi

- Scrivere un programma che scrive a terminale
  - la radice quadrata di 2, il cubo di 2 il seno di pigreco/4
- Scrivere un programma che:
  - controlli se uno o più argomenti opzionali sono passati al main al momento dell'avvio (e in caso contrario, avvisi l'utente)
  - scriva l'intera lista di argomenti opzionali passati al main
  - chieda all'utente di inserire un intero a scelta tra 1 e 2, e restituisca a terminale il valore inserito, o un messaggio di errore in caso di inserimento di altri interi
- Scrivere un programma che, dato un array di n interi casuali, lo ordini dal più piccolo al più grande (suggerimento: per generare i numeri casuali, usate la funzione *rand()* contenuta in *cstdlib*)

I puntatori

- Un **puntatore** è una variabile che contiene un indirizzo di memoria




```
double pi_greco = 3.1415 ;

pi_greco_ptr = &pi_greco ;
```

```
double * pi_greco_ptr ;

std::cout << *pi_greco_ptr << "\n" ;
```

- se **x** contiene l'indirizzo di **y**, si dice che x *punta* ad y

Gli operatori * e &

- Con i puntatori si utilizzano due operatori speciali, ***** e **&**.
- L'**operatore &** ritorna l'indirizzo della variabile che precede. Tradotto verbalmente, suonerebbe come *"l'indirizzo di"*.

```
int value;
int *p;

p = &value;
```

- Scrivo nella variabile puntatore *p* il valore dell'indirizzo della variabile *value*
- Verbalmente *"p riceve l'indirizzo di value"*

- L'**operatore** * ritorna il valore della variabile che si trova all'indirizzo specificato dal suo operando. Verbalmente *"all'indirizzo"*.

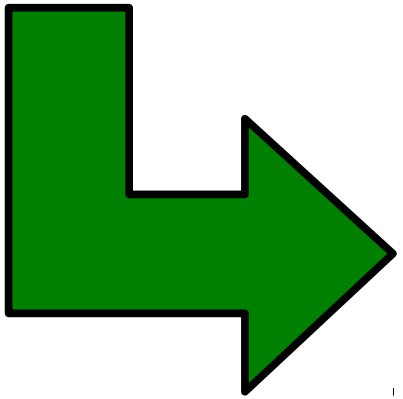
```
int value;
int num;
int *p;

value = 1000;
p = &value;
num = *p;
```

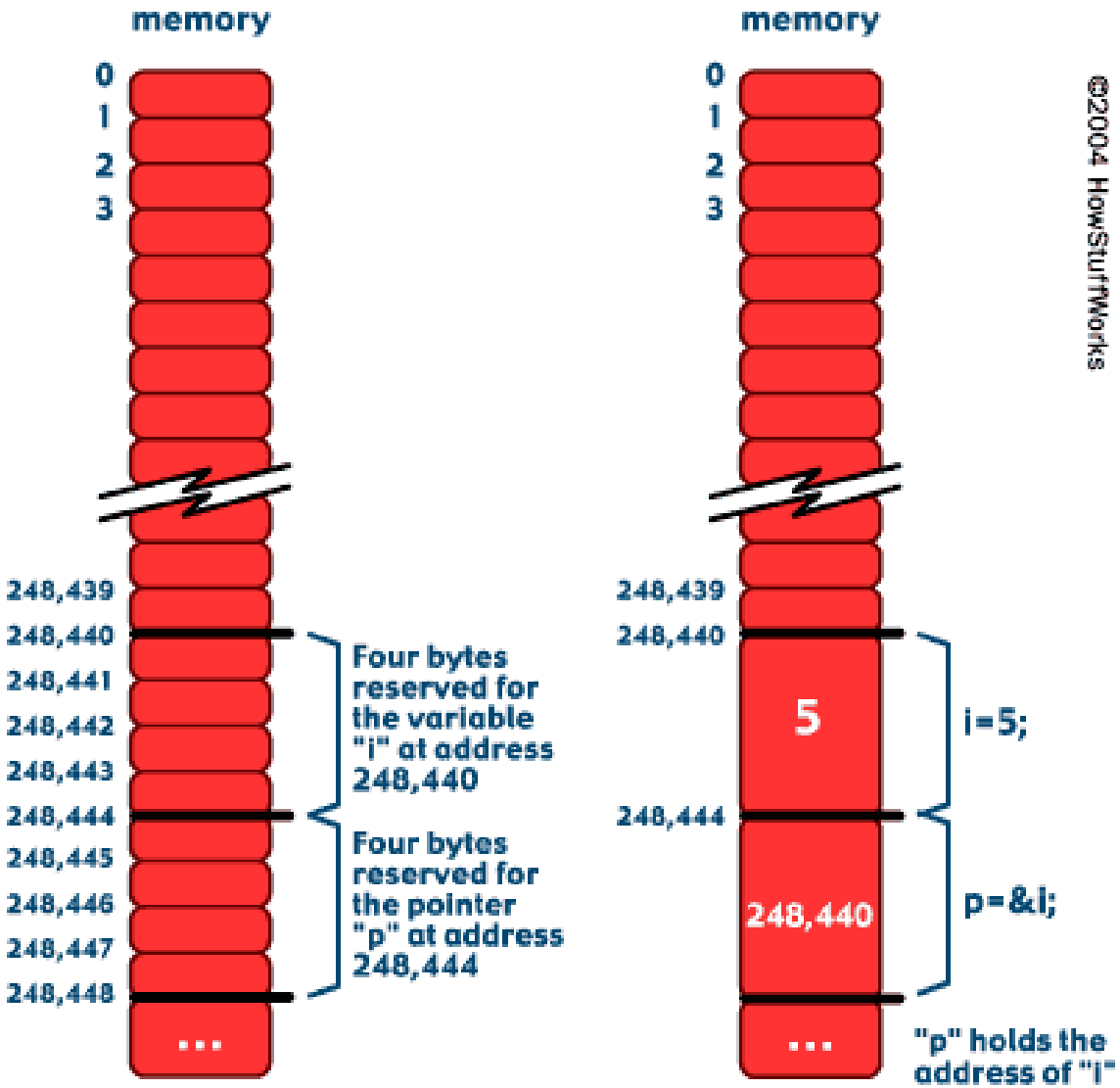
- *"num riceve il valore che si trova all'indirizzo puntato da p"*
- Ora il valore di num è 1000

Puntatori e RAM

Vediamo che cosa accade nella RAM del calcolatore

```
int i ;
i = 5 ;
int * p ;
p = &i ;
```





Puntatori e referenze

- NB: Anche i puntatori sono variabili e possono cambiare valore

```
double pi_greco = 3.1415 ;
double * altro_pointer ;
altro_pointer = &pi_greco ;
double nepero = 2.7183 ;
altro_pointer = &nepero ;
```

- Un puntatore si può creare senza assegnargli un valore
- Il valore del puntatore e' l'indirizzo di memoria della variabile alla quale punta

- Le referenze invece si comportano come puntatori, ma sono costruite su una variabile specifica e rimangono vincolate ad essa

```
double pi_greco = 3.1415 ;
double & pi_greco_ref = pi_greco ;
std::cout << pi_greco_ref << "\n" ;
```

- Una referenza si crea a partire da una variabile esistente
- Si utilizza come una variabile e si comporta come un puntatore

Gestione della memoria

- Solitamente il C++ decide come gestire la memoria
- Se vogliamo farlo noi possiamo utilizzare l'operatore **new**

```
double * pointer = new double (1.5) ;
std::cout << *pointer << std::endl ;
delete pointer ;
```

Il valore 1.5 viene assegnato alla variabile a cui punta il puntatore

Dopo averla utilizzata, dobbiamo liberare la memoria con **delete**

Il puntatore e' l'unico legame con la variabile!

- Si possono creare anche vettori con **new**
- Questo ci permette di scegliere **runtime** la dimensione del vettore

```
double * array = new double [10] ;
for (int i=0 ; i<10 ; ++i)
  {
    array[i] = 0.31 * i ;
  }
delete [] array ;
```

Il valore degli elementi va assegnato dopo la definizione

Anche in questo caso bisogna chiamare il **delete[]**

UN ARRAY E' UN PUNTATORE!

I puntatori - esercizi

- Scrivete un programma che assegni il valore di una variabile ad un'altra utilizzando un puntatore. Fatevi inoltre stampare a terminale i valori e gli indirizzi di ogni variabile prima e dopo l'assegnazione.

- Dichiarate un puntatore e poi cercate di assegnargli direttamente un valore numerico. Cosa succede? Perchè?

- Utilizzate **new** e **delete** per creare e distruggere una variabile double ed un array di double

Le funzioni

- Serie circoscritte di istruzioni possono essere isolate in funzioni

```
double raddoppia (double input)
{
  return input * 2 ;
}
```

- La funzione si definisce prima del programma principale (main)
- Puo' avere diversi oggetti come input ed (al piu') un solo oggetto output
- All'interno del programma principale viene chiamata secondo il prototipo dichiarato

```
double valore_iniziale = 4 ;
double valore_finale = raddoppia (valore_iniziale) ;
std::cout << valore_iniziale
          << " x 2 = "
          << valore_finale << "\n" ;
```

Il passaggio di argomenti

- La stessa funzione può essere implementata in diversi modi e le variabili possono esserle passate in diversi modi

```
double raddoppia (double input)
{
  return input * 2 ;
}
```

```
double raddoppiaReference (double & input)
{
  input = input * 2 ;
  return input ;
}
```

```
double raddoppia2 (double input)
{
  input = input * 2 ;
  return input ;
}
```

```
double raddoppiaPointer (double * input)
{
  *input = *input * 2 ;
  return *input ;
}
```

```
raddoppia:          4 x 2 = 8
raddoppia2:         4 x 2 = 8
raddoppiaPointer:   8 x 2 = 8
raddoppiaReference: 8 x 2 = 8
```

- Non tutti gli output sono uguali, come mai?

Divisione in diversi file

- Per evitare di avere programmi troppo lunghi e per semplificare la vita al compilatore, di solito le funzioni sono impacchettate in librerie

esempio06.h

```
#ifndef esempio06_h
#define esempio06_h

double raddoppia (double input) ;

#endif
```

Contiene le definizioni di variabili e funzioni (i prototipi)

Le istruzioni **#ifndef, #define, #endif** servono per evitare di duplicare le definizioni se il file viene usato piu' volte

esempio06.cc

```
#include "esempio06.h"

double raddoppia (double input)
{
  return input * 2 ;
}
```

Contiene l'implementazione delle funzioni

Conosce il prototipo da esempio06.h attraverso l'istruzione **#include** (che incolla il contenuto di esempio06.h dove e' chiamata)

Uso nel programma principale

- Nel programma principale, le funzioni definite nelle librerie sono utilizzate senza bisogno di implementarle

```
#include<iostream>
#include "esempio06.h"

int main (int numArg, char *listArg[])
{
  double valore_iniziale = 4 ;
  double valore_finale = raddoppia (valore_iniziale) ;
  std::cout << "raddoppia:          "
            << valore_iniziale
            << " x 2 = "
            << valore_finale << "\n" ;
  return 0 ;
}
```

La grammatica e' nota dall'inclusione di esempio06.h

L'implementazione viene linkata automaticamente dal c++, che viene chiamato così:

```
c++ -o esempio06 esempio06.cc esempio06.cpp
```

Le stringhe in C e C++

- Le parole sono gestite da vettori di lettere in C

```
char parola[12] = "pippo" ;
std::cout << parola << std::endl ;
int num3 = 5 ;
sprintf (parola,"pippo_%d", num3) ;
std::cout << parola << std::endl ;
```

La dimensione della stringa deve essere scelta con cautela

La funzione **sprintf** permette di cambiare il contenuto della stringa (comprese altre variabili)

- In C++ esiste un tipo dedicato, std::string
- Per questo e' necessario includere la libreria: `#include<string>`

```
std::string parolaBis ;
parolaBis = "pippoBis" ;
std::cout << parolaBis << std::endl ;
parolaBis += "_aggiungo_" ;
std::cout << parolaBis << std::endl ;
```

Non e' necessario scegliere la dimensione della stringa

Si possono aggiungere contenuti alla stringa con l'operatore +

Ordine!

- **SINTASSI** = insieme di sane abitudini, oltre alla scrittura corretta del codice. Ad esempio:
  - **indentare** il codice
  - **commentare** il codice (per se stessi e per gli altri)
  - scegliere un **modo coerente** di scrivere il codice (parentesi graffe a capo o non a capo, spaziature...)
  - scegliere **nomi lunghi** ed **esplicativi** per le variabili

Esercizi

- Scrivere una funzione che calcoli il fattoriale di un numero intero non negativo
- Scrivere la funzione fattoriale in modo ricorsivo, cioè facendo in modo che la funzione che calcola il fattoriale chiami se stessa

  Inoltre, visto che le cose si imparano bene solo quando ci si sbatte la testa, prendete gli esempi di questa lezione e provateli sui vostri pc!
- Fatevi stampare l'input del main
- Provate tutte le strutture di controllo (if, switch, for, while)
- **FATE** ginnastica con array, pointers e referenze
- Provate a passare argomenti alle funzioni by ref e by ptr
- Dividete il vostro codice in più file (".h",".cc",".cpp") e compilatelo